Early Performance Results on the NRL SGI Altix 3000 Computer

Wendell Anderson and Robert Rosenberg
Naval Research Laboratory (NRL-SSC), Stennis Space Center, MS
{Wendell.Anderson, Robert.Rosenberg}@nrl.navy.mil

Marco Lanzagorta
Scientific & Engineering Solutions, Stennis Space Center, MS
Marco.Lanzagorta@nrl.navy.mil

Abstract

Since December 2002 the Naval Research Laboratory (NRL) has been evaluating an Altix 3000, the newest high performance computing system available from Silicon Graphics, Inc. (SGI). The Altix is a departure from the previous products from SGI in that instead of using MIPS processors and the IRIX operating system, the Altix uses the Intel Itanium IA-64 processor and SGI ProPack (based on Linux Red Hat) operating system. The Altix still has the brick concept of system configuration and the SGI NUMAlink for the inter-module network for the shared memory. The Altix runs under a single image of the operating system and supports parallel programming through OpenMP, MPI, CoArray, FORTRAN, and an automatic parallelizing compiler. Various codes have been evaluated with respect to their ease of portability and their performance on the Altix as compared to other high performance computers.

1. Introduction

NRL began its evaluation of the SGI Altix with the delivery in December 2002 of a system with sixteen 900 MHz Itanium II processors and 16 gigabytes of memory. The machine was later upgraded to 1000 MHz Itanium II processors. This early access provided NRL with a machine for the development and testing of Kerberos and the connection to the Altix via Asynchronous Transfer Mode (ATM). In April 2003, an additional 112 processors were shipped to NRL and the memory expanded to 256 gigabytes. The system then consisted of 32 C-bricks, 18 R-bricks, 3 IX-bricks, and 4 PX-bricks. Each C-brick contained 4 processors and 8 gigabytes of memory and the R-bricks contain routers for connecting the C-brick memories. Each of the PX bricks has 12 PCI-X slots. Currently the PX bricks have four ATM cards, two fibre channel cards, an infiniband card, and a scsi card. Each of the IX bricks has 12 PCI-X slots and an IO9 card (the base IO).

Using alpha software, NRL started operating the Altix as a single image 128 processor system, the first site outside of SGI to run a system that large as a single image. The system was accepted and opened up to early access users in July 2003. At this time, the Altix was running OpenAFS 1.2.11 natively letting the user's home directory on the Altix be the same as their home on the other HPC resources at NRL. Connections to the network were via HE 622 ATM cards and an additional 339 gigabytes of high speed disk space was available for the direct I/O of data to the Altix. Users could submit their jobs via the PBS Pro 5.1 queuing system and the Intel Fortran and C/C++ compilers were available for compiling and linking users codes. In August, the processors were upgraded to 1.3 GHz Itanium II processors.

2. Problems and Methodology

In order to evaluate the machine, hardware and software problems were tracked and the response of SGI and Intel noted. In addition, the experiences of users and NRL staff in developing, porting, and running software were followed. Over the course of the evaluation three different Intel FORTRAN and C/C++ compilers have been used: Versions 7.0, 7.1, and 8.0. Each compiler version had its advantages and disadvantages.

Five codes (Flux, Causal, NRLMOL, TEMPO_8, and Lanczos) that represent a cross section of applications and parallel programming paradigms were ported to the Altix by NRL staff and performance comparisons were made with other HPC systems.

The Flux code[1] calculates temperature dependent electronic spectra and super conducting densities. The F90 code interactively updates the values at each point in a 4-D grid and performs Fast Fourier Transformation (FFT's) over each of the axis of the grid. Two versions of

the program exist: one using the zzfft routines from the SGI scientific library to perform FFTs and the other using the FORTRAN numerical recipes FFT. Most of the parallelization is obtained by using the automatic parallelization capabilities of the compiler, although there are a few places where OpenMP directives are used to execute in parallel subroutines called within DO loops.

The Causal code[2] models the propagation of an acoustic wave in water via a finite difference time domain (FDTD) representation of the linear wave equation that takes into account the dispersive properties of the medium by the addition of the derivative of the convolution between an operator and the acoustic pressure. The parallelization of this FORTRAN code is done entirely through OpenMP directives applied to the loops that update the acoustic wave at each grid point.

The NRLMOL code[3] implements the Density-Functional formalism for clusters and molecules. NRLMOL is a FORTRAN code that uses MPI to parallelize the problem by using a master process to distribute work to slaves. All parallelism is carried out through this master/slave relationship.

The TEMPO_8 code[4] uses a genetic algorithm to create a set of rules for the allocation of resources in the development of weapons and defenses. The computations of the problem are set up as a data flow graph and the Processing Graph Method Tool (PGMT) is used to convert the data graph to C++ code that assigns a set of nodes to each processor and uses MPI to move data between nodes. PGMT hides from the user all of the details required in setting up and synchronizing the MPI calls.

The Lanczos code[5] arose from the study of the low temperature thermodynamic properties of many body quantum systems. Most of the work performed in the code is in the multiplication of a sparse matrix by a dense vector. Two versions of this code exist: a C++ version and a F90 version. The C++ version distributes the sparse array across processors with each processor getting a set of rows and the dense vector moved between processors by using MPI. The F90 version relies on a compiler switch to automatically parallelize the code.

3. Results

3.1. Hardware.

Hardware problems with the machine have been relatively few. During the first twelve months of operation, the only failures were two of the voltage regulator modules (VRM), a Dual Inline Memory Module (DIMM) from the original 16 processor system, a C-brick, and one of the drives in the scratch disk system. When the original 128-processor system was first booted, only 126 processors were recognized by the operating system. Initially SGI personnel were not sure if the problem was software or hardware. The problem was finally traced to a corrupted Programmable Read Only Memory (PROM) and when this was replaced the system booted with all 128 processors.

When the C-brick failed, reducing the Altix to a 124-processor system, the problem took almost a month to repair as SGI tried replacing various parts of the C-brick. By the time the problem was finally resolved, maintenance personnel had replaced almost every part on the C-brick. The repair time would have been much quicker if SGI had a procedure for just swapping out the C-brick.

3.2. Software.

The alpha and beta operating system software has been remarkably stable with only occasional system hangs. One problem that has occurred several times has been the hanging of a user process in a state where any UNIX command (like ps and top) that accessed the status of that program would not respond. The process could only be cleared from the system by rebooting the Altix. For the past four months, NRL has been running with the latest PROMs and the released version 2.4 of SGI ProPack. In that time, no hangs of the system have been observed.

The Message Passing Interface (MPI) is implemented on the Altix via the Message Passing Toolkit (MPT). MPT is fully compliant with the MPI 1.2 specification and contains a number of MPI-2 features. In order to build an MPI program, the user merely needs to include the proper MPI definitions file in the source code, and specify `-lmpi` when linking in order to include the library containing the MPI routines. To run the executable program using n processors the command is simply

```
mpirun -np n program.exe
```

Users with Fortran MPI codes with little inter-processor communications have generally run 1.5–2 times faster on the Altix than the Origin at NRL. Those porting codes from Origin systems reported only one problem associated with MPI—the use of the `MPI_win_create` routine to map memory to a window for MPI-2 one-sided communications. For both the Origin and Altix, the window must be in remotely accessible memory. However, the two systems make different decisions when allocating memory. A FORTRAN allocatable array works on the Origin for `MPI_win_create`, but does not on the Altix. The current work-around is the use of a fixed dimension array rather than the allocatable array used in the original code.

The Altix also supports the OpenMP standard. Porting OpenMP codes to the Altix have been straightforward. Compiling and linking OpenMP codes requires the addition of the –openmp switch to the command line. Only one user has reported a problem related to OpenMP when porting code. In this case the code uses USE modules to pass data between routines. When using data from these arrays as private variables within OpenMP DO sections, the program produces incorrect results. This problem is currently being researched by SGI. In the meantime, a temporary workaround based on storing multiple copies of the n-dimensional array in an n+1 dimensional shared array is being used.

Like the MIPSPro compilers on the Origin 3800, the Altix supports the automatic parallelization of codes (via the `-parallel` switch for the Intel compiler). OpenMP and automatic parallelization can be supported within the same program but not in the same file. If both the `-openmp` and `-parallel` switches are specified when compiling a file, then for any files that contain OpenMP directives the `-parallel` switch is ignored. Porting codes from the Origin that use both methods in the same file requires that either all parallelization be performed using OpenMP or that the OpenMP portions of the file be moved into a separate file. This can entail a significant amount of work for programs with thousands of lines of code and a moderate use of OpenMP directives.

The experience of the NRL staff with codes compiled and run on both the Origin and the Altix is that the Intel compilers are not nearly as good as the MIPSPro in producing efficient code, especially for auto parallelization. Problems seen included slow versions of some of the intrinsic functions, not combining a set of nested DO loops with the same starts, stops, and increments into a single nested DO loop, not moving unnecessary calculations out of inner loops, failing to parallize code that obviously should be, and not reordering loops to obtain improved performance.

Improvements in the version 8 FORTRAN compiler produced code that ran much faster than those from version 7. However, the new version of the compiler has two main drawbacks. Numerous bugs had crept into the new compiler and many codes no longer compiled. With the auto parallelization turned on, fewer DO loops would automatically parallelize. One code went from parallelizing 410 loops to parallelizing 18. Intel did provide an undocumented switch

```
-mPAROPT_auto_parafter_hlo=FALSE
```

that provided better parallelization. Use of this switch increased the number of DO loops parallelized to 139.

NRL has also installed the CoArray Fortran compiler available from Rice University and verified the compiler could build and execute the test codes provided. The main difficulty in installing CoArray was obtaining the correct versions of the additional software (Automake, Autoconf, binutils, gcc 3.3.2, ecc 7.1, and efc 7.1) required for the installation. CoArray FORTRAN currently only works with version 7 of the Intel compilers.

While the Origin 3800 had two versions of the SGI math libraries, the older COMPLIB.SGIMATH and the newer SCSL, the Altix has only the latter. The main incompatibilities with the new library of codes that were developed for the old library were the names and parameters of the FFT routines. Users who had not updated their codes to use the FFTs in the SCSL library have had to modify their codes to use the newer calling sequences.

Some users whose codes used FFTs from the SCSL library found that codes that ran fine on the Origin died when run on the Altix. The man page for the INTRO_FFT for IRIX 6.5 states that

"The second area of difference is the isys parameter array.... On any implementation, you can use the default values by using an argument value of 0. In the Origin series implementation, isys(0)=0 and isys(0)=1 are supported"

Users of the Altix whose codes on the Origin used a default argument of 0 have found that these codes no longer work on the Altix. Instead isys must be an array. Users who defined the isys parameter as an array have reported no problems porting their code to the Altix.

Next the NRL staff examined the performance of five codes that had been ported from the Origin to the Altix - Flux, Causal, NRLMOL, TEMPO_8, and Lanczos.

3.3. Flux.

Flux is a legacy code written originally for the Connection Machine and ported over the years to SGI Origins and the Cray Multi-threaded Architecture (MTA-2). Little effort (outside of the FFT portions of the code) had been made to optimize the code for machines with cache. Parallelization of the code relied heavily on applying the automatic parallelization switch of the Fortran compiler to the hundreds of DO loops in the code. Porting the code to run properly on the Altix required only changes to the compiler switches. Timing results for the Flux code and the FFT code within it for the SGI Origin and Altix systems at NRL are given in Tables 1 and 2.

Table 1. Flux Wall Click Times (secs)

	Origin	Altix
1 processor	194	199
8 processors	42	161

Table 2. FFT Times (secs)

	Origin	Altix
1 processor	74.2	26.4
8 processors	12.9	6.5

While the FFT times on the Altix were impressive, the overall performance of the code has been quite disappointing, especially with respect to scalability to multiple processors. Efforts are currently underway to modify the code to improve performance.

3.4. Causal.

The porting of causal from the Origin to the Altix merely required the changing of the switches used for compiling and linking. Results of running the original code on the Origin and Altix are contained in Table 3.

Table 3. Causal Original Code Time (secs)

	Origin	Altix
1 processor	49170	13013
40 processors	1396	302

The code ran approximately four times faster on the Altix for both the single and forty processor cases. Scaling was actually super-linear on the Altix. Impressive as these results were, examination of the code revealed that the inner most loop of the triply nested DO loop where most of the calculations were performed involved a call to the intrinsic mod function. With a little reprogramming, the mod was moved entirely outside of the nested DO's. The timing results for the modified code are given in Table 4.

Table 4. Causal Modified Code Time (secs)

	Origin	Altix
1 processor	27445	4485
40 processors	1182	114

While the new code ran twice as fast on the Origin for 1 processor, the speed up was only 20% for the code on forty processors. On the other hand, the new code ran almost three times faster than the old code for both cases on the Altix.

3.5. NRLMOL.

As in the previous case the conversion of the NRLMOL code required only changes to the compiler switch. This was expected as this version of NRLMOL was the result of a CHSSI project where code portability is a high priority and the code had been run on a multitude of high performance computers. In addition to the Origin and the Altix, the code was also run on the Linux Networx cluster at the ARL MSRC. Two cases were examined – a medium case (Man-12 Acetate) run on 1 and 48 processors and a large case (Fe-TACN molecule) run on 16 and 64 processors. Timings on the three machines are given in Table 5 and Table 6.

Table 5. NRLMOL Medium Timings (secs)

	Origin	Altix	Linux Networx
1 processor	6019	4421	1650
48 processors	275	193	94

Table 6. NRLMOL Large Timings (secs)

	Origin	Altix	Linux Networx
16 processor	58713	46011	28262
64 processors	22041	15082	18076

For the medium size case the Altix ran about 30% faster than the Origin but took two to three times longer than on the Linux Networx. For the larger case the Altix showed a 20% and 30% improvement over the Origin. For 16 processes the LinuxNetwork was about 40% faster. The big surprise was that on 64 processors the Altix was actually faster than the Linux Networx box. The reasons for this behavior are still being investigated.

3.6. TEMPO_8.

The TEMPO_8 code and the associated PGMT libraries were originally developed on an Origin using g++ 2.95.2. Later the code was converted to compile and run with g++ 3.2.2. This version was then ported to the Altix. Only one set of errors was found in the PGMT libraries that associated with the difference in default byte lengths between a 32 bit machine and a 64 bit machine. All of the code automatically generated by PGMT for the TEMPO_8 application compiled with no problems. Running with population sizes of the order of 1000, running times were 25% faster on the Origin than the Altix. Finally the code was compiled using the Intel icc compiler. No errors were found, but the compiler did generate hundreds of warning messages. An improvement of about 8% in running time over the g++ compiler was found.

3.7. Lanczos.

The Lanczos code has been run on a multitude of platforms over the last few years and typical timings for generating 1000 Lanczos coefficients for a very large

complex sparse matrix (a 10,000,000 by 10,000,000 with an average of 33 non-zero elements per row) are given in Table 7. The first seven entries are for the C++/MPI version of the machine and the last four for a F90 version of the code. The number of processors on the X1 is the number of SSP's used. This code is not well suited for cache type machines as the reuse of data is minimal. MPI versions of the code require that much of the dense vector be moved between iterations degrading the performance relative to the F90 versions.

Table 7. Lanczos Wall Clock Times

Platform	Speed MHz	16P mins
Cray X1	800	212
SGI Origin	400	158
IBM P3	375	143
SGI Altix	1300	113
Intel Pentium III	933	103
COMPAQ ES45	1000	90
Linux Networx	3060	69
SGI Altix	1300	110
SGI Origin	500	90
Cray X1	833	28
Cray MTA-2	220	23

4. Significance to DoD

As the demands for high performance computing grow with increased demands for computations, memory, and I/O, the Department of Defense must continue to provide the best HPC resources to its personnel. In order to maintain and expand its capability, given the large range of applications, that DoD must support, one architecture will not be able to satisfy, in a cost effective way all of DoD's HPC needs. The Altix with its balanced architecture of processor speed, memory, and I/O will be needed for those problems that need to achieve a T3 (a teraflop of computations, a terabyte of memory, and a terabit per second of I/O) type of performance. In order to meet the needs of such programs, NRL has just added through the auspices of the HPCMP, a second Altix system with 256 processors and a terabyte of memory to its center. In the next few months, the system will be upgraded by the addition of visualization software and additional memory bandwidth to support ongoing research programs

5. Systems Used

Computations were performed on resources provided by the DoD HPCMPO. Calculations were performed on the IBM P3 and COMPAQ ES45 at the Aeronautical Systems Center (ASC), the SGI Origin 3800 the US Army Engineer Research and Development Center (ERDC), the Linux Networx cluster at the Army Research Laboratory (ARL), and the SGI Altix 3000, SGI Origin 3800, and MTA at the Naval Research Laboratory (NRL) shared resource centers.

6. CTA

The codes used in the evaluation of Altix come from a broad spectrum of the HPC CTA's including CCM, COM, CWO, and FSM.

References

The original codes were written by Dr. Daryl Hess (Flux), Dr. Guy Norton (Causal), Dr. Mark Pederson (NRLMOL), and Dr. Steve Hellberg (Lanczos), all while at NRL and Prof. Rodney Johnson (TEMPO_8) of the Naval Post Graduate School. The authors are deeply appreciative to all of the developers and to the SGI staff, especially Bryan Bush, David Anderson and C.J. Schuyta for their support and willingness to answer questions.

1. Diesz, J., D. Hess, and J. Serene, "Phase Diagram for the attractive Hubbard Model in two dimensions in a conserving approximation." *Physical Review B*, 66, 2002.

2. Norton, G. and J. Novarini, "Including dispersion and attenuation directly in the time domain for wave propagation in isotropic media." *J. Acoust. Soc. Am.*, 113, 2003, p. 3024.

3. M. Pederson, D. Porezag, J. Kortus, and D. Patton, "Strategies for massively parallel local-orbital-based electronic structure calculations." *Physica Status Solidi B*, 217, 2000, p. 197.

4. Johnson, R., Z. Michalewicz, M. Melich, and M. Schmidt, "Coevolutionary "TEMPO" Game." *2004 Congress on Evolutionary Computation*. Portland, OR, June 2004.

5. Anderson, W., M. Lanzagorta, and C. Hellberg, "Analyzing Quantum Systems Using the Cray MTA-2." *Proceedings of the Cray Users Group*, Columbus, OH, May 2003.